Procesadores de Lenguajes

Entrega 1

María Arranz Lobo
Celia Rubio Madrigal

23 de abril de 2021

```
1  import quicksort from sort. € Imports.
2
3  type matriz = arr\arr\ent. € Tipos con nombre.
4
5  const ent pi = 3. € Constantes.
6
7  € Registros con parámetros.
8  data coleccion (ent size) = {
9    arr\ent cosas = (size,0).
10   bul vacio = size==0.
11 } .
12
13 € Funciones.
14 function my_quick return arr\ent (arr\ent v) {
15     arr\ent w = quicksort(v).
16     return w.
17 }
18
19 € Siempre al final.
20 main {
21     ent a = kin().
22     while (a > 0) {
23         kut(a).
24         a = a - 1.
25     }
26
```

```
27
28      arr\ent v = [2 * 3, 9 / 3, 1 + (-1)].
29      kut(v ! 3).
30
31      arr\ent w = my_quick(v).
32      for(e : w) {
33          kut(e).
34      }
35
36      coleccion my_bolsa(3).
37
38      if (my_bolsa->vacio) {
39          kut(-1).
40      }
41      else {
42          kut(my_bolsa->cosas ! 0).
43      }
44  }
```

El archivo siempre debe tener un `main` al final que es el que se ejecutará. Encima de él podrán ir imports (todos al comienzo), declaración de registros, de funciones o de tipos definidos, variables globales o constantes.

Los comentarios son con el símbolo €.

1 Identificadores y ámbitos de definición

Las instrucciones acaban en punto. El ámbito de definición está delimitado por los corchetes.

Declaración de tipos básicos, siempre con valor inicial. Los identificadores solo pueden tener letras minúsculas, números y barras bajas (ambos no al comienzo). Se prefiere el estilo de *snake_case*.

```
ent taxicab_number = 1729.
```

Declaración de registros, fuera del `main`. Como son una declaración también acaban en punto.

```
data coleccion () = {
  arr\ent cosas = [0].
  bul vacio = no.
} .
```

Se accede a sus componentes mediante ->.

```
coleccion bolsa().
bolsa->cosas ! 0 = 1.
```

Pueden llevar parámetros de inicialización entre paréntesis, pero deberán ser constantes.

```
data coleccion (const ent size) = {
  arr\ent cosas = (size,0).
  bul vacio = size==0.
} .
main {
    coleccion my_bolsa(3).
}
```

Declaración de funciones. Los argumentos se pasan por valor.

```
function identificador return tipo (params) { cuerpo }
```

Hay alguna restricción sobre los tipos de los parámetros y/o resultado de las funciones?

Imports. Se sobrescriben en orden, así que no se puede acceder a funciones con el mismo nombre de distintos imports. Van al comienzo del archivo.

```
import quicksort from sort.
```

Hay alguna restricción sobre dónde se puede llamar a las funciones?

El resto de declaraciones van en orden indistinto, salvo el `main` al final.

2 Tipos

Tipos básicos (y entrada - `kin`/salida - `kut`, solo definidos para ellos).

```
bul x = si.
bul y = kin().
ent a = 0.
kut(a+1).
car c = 'f'.
```

Expresiones enteras:

```
ent patata1 = my_rec->a * my_rec-> b + my_arr !
   my_rec->c.
ent patata2 = ((my_rec->a) * (my_rec->b)) + (my_arr !
   (my_rec->c)).
€ Son iguales.

ent cordero1 = my_arr ! my_rec->a + my_rec->b.
ent cordero2 = my_arr ! (my_rec->a + my_rec->b).
€ No son iguales.
```

Expresiones buleanas:

```
bul var1 = b1==b2.
bul var2 = 3+2==5.
bul var3 = 3 > 1+1==si.

bul var4 = 3 > 1+1 & ¬no | 1+1 != 2.
bul var5 = ((3 > (1+1)) & (¬no)) | ((1+1) != 2).
€ Son iguales.
```

Arrays. Se accede a sus elementos mediante el operador exclamación. Tienen longitud fija, que se consulta con la palabra reservada `length`.

```
arr\ent v1 = [1,2,3].

arr\ent v2 = (3,0).
€ Par entero (longitud), tipo (inicialización).

arr\arr\ent m1 = [[1,1,1],v1,v2].
€ Matrices multidim con longitud consistente.
kut(length(m1 ! 0)).

arr\arr\ent m2 = (3,[0,0,0]).
arr\arr\ent m3 = (3,(3,0)).
€ m2==m3.
```

Tipos con nombre. El usuario también puede definir tipos usando `type`.

```
type matriz = arr\arr\ent.
```

Operadores con prioridades y asociatividades.

Op.	Tipo	Prioridad	Asociatividad	Semántica
=	Binario infijo	8	No	Asignación
!=	Binario infijo	7	Izquierda	Distinto
==	Binario infijo	7	Izquierda	Igual
|	Binario infijo	6	Izquierda	Or
&	Binario infijo	5	Izquierda	And
<	Binario infijo	5	Izquierda	Menor
<=	Binario infijo	5	Izquierda	Menor o igual
>	Binario infijo	5	Izquierda	Mayor
>=	Binario infijo	5	Izquierda	Mayor o igual
¬	Unario prefijo	4	No	Not
+	Binario infijo	4	Izquierda	Suma
−	Binario infijo	4	Izquierda	Resta
∗	Binario infijo	3	Izquierda	Multiplicación
/	Binario infijo	3	Izquierda	División entera
%	Binario infijo	3	Izquierda	Módulo
−	Unario prefijo	2	No	Opuesto
!	Binario infijo	1	Izquierda	Índice array
−>	Binario infijo	0	Izquierda	Acceso registros

Se pueden definir constantes a través de la palabra reservada `const`.

```
€ Estructura definición de constantes:.
const tipo nombre = valor .

const matriz unitaria = [[1,0],[0,1]].

const ent max = 100.
```

3 Conjunto de instrucciones del lenguaje

Asignaciones. El operador de asignación es el igual, la estructura básica de asignación es, por tanto la siguiente.

```
ent my_num = 0.
arr\ent my_arr = [1,2,3].
my_arr ! 2 = 0 .
```

Bucles. Existe un while, y un for que solo itera elementos de arrays mediante paso de valor.

```
ent a = kin().
while (a > 0) {
    kut(a).
    a = a - 1.
}

arr\ent w = my_quick(v).
for(e : w) {
    kut(e).
}
```

Condicionales. La definición de ramas de ejecución condicional se hará a través de las estructuras if (1 rama) o if-else (2 ramas). De manera que al tener la posibilidad de anidamiento permitirán definir la totalidad de casos que sea necesario.

```
€ Calcular el valor absoluto de un número.
if (num < 0) {
    num = -1 * num.
}
```

```
€ Determinar paridad de un número.
if (num % 2==0) {
    par = si.
}
else {
    par = no.
}
```

4 Más ejemplos

Función que calcula el máximo de un array de enteros.

```
function find_max return ent (arr\ent array) {
    ent max = array ! 0.
    for(elem : array) {
        if (elem > max) {
            max = elem .
        }
    }
    return max .
}
```

Programa que dadas dos matrices 2x2 introducidas por el usuario comprueba que son inversas una de la otra.

```
1  type matriz = arr\arr\ent.
2  const matriz unitaria = [[1,0],[0,1]].
3
4  function producto return matriz (matriz a, matriz b){
5      ent c1 = a!0!0 * b!0!0 + a!0!1 * b!1!0.
6      ent c2 = a!0!0 * b!0!1 + a!0!1 * b!1!1.
7      ent c3 = a!1!0 * b!0!0 + a!1!1 * b!1!0.
8      ent c4 = a!1!0 * b!0!1 + a!1!1 * b!1!1.
9
10     return [[c1,c2],[c3,c4]].
11 }
12
13 function inversa return bul (matriz a, matriz b){
14     return producto(a,b)==unitaria.
15 }
16
17 function lee_matriz return matriz() {
18     ent a1 = kin().
```

```
    ent a2 = kin().
    ent a3 = kin().
    ent a4 = kin().
    return [[a1,a2],[a3,a4]].
}

main {
    matriz a = lee_matriz().
    matriz b = lee_matriz().
    bul sol = inversa(a,b).
    if (sol) {
        kut (si).
    }
    else {
        kut (no).
    }
}
```

Ejemplo de archivo a importar para facilitar el uso de arrays de caracteres.

```
type string = arr\car.
const ent max_string = 100.

function copy return string (string source, ent
   longitud) {
    string dest = (longitud, ' ').
    i = 0.
    while (i < longitud) {
        dest!i = source!i.
        i = i + 1.
    }
    return dest.
}

function kin_string return string (){
    car letra = kin().
    string aux = (max_string,' ').
    ent i = 0.
    while (letra != '\n' & i < max_string) {
        aux!i = letra.
        letra = kin().
        i = i + 1.
    }
    if (i < max_string) {
        aux!i = letra.
```

```
25          i = i + 1.
26      }
27      return copy(aux,i).
28  }
29
30  function kut_string return ent (string cadena){
31      for(letra : cadena) {
32          kut(letra).
33      }
34      return 0.
35  }
36
37  main {}
```